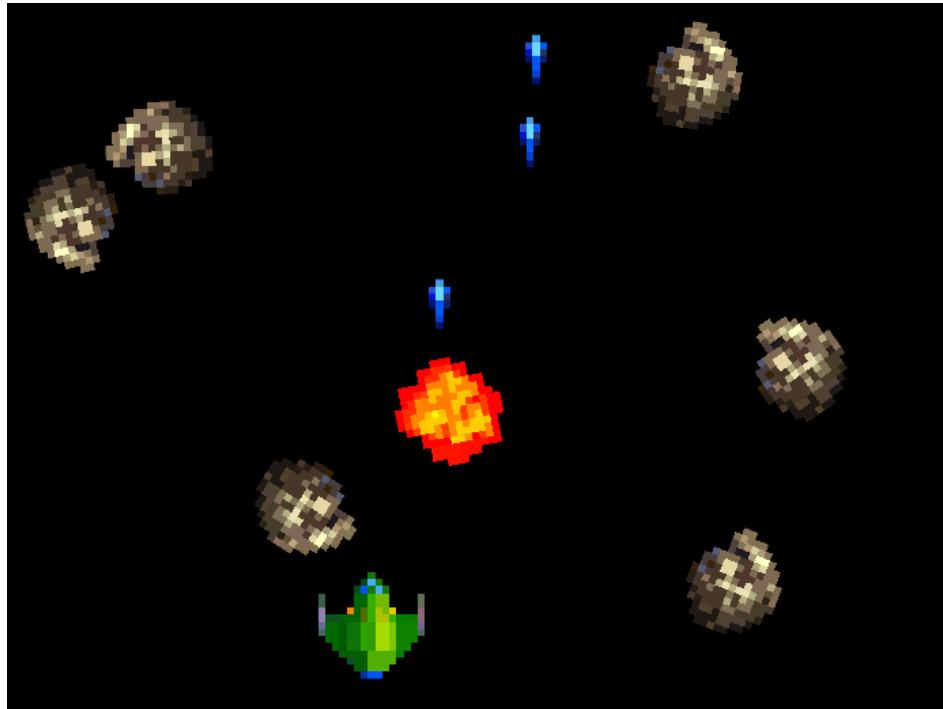
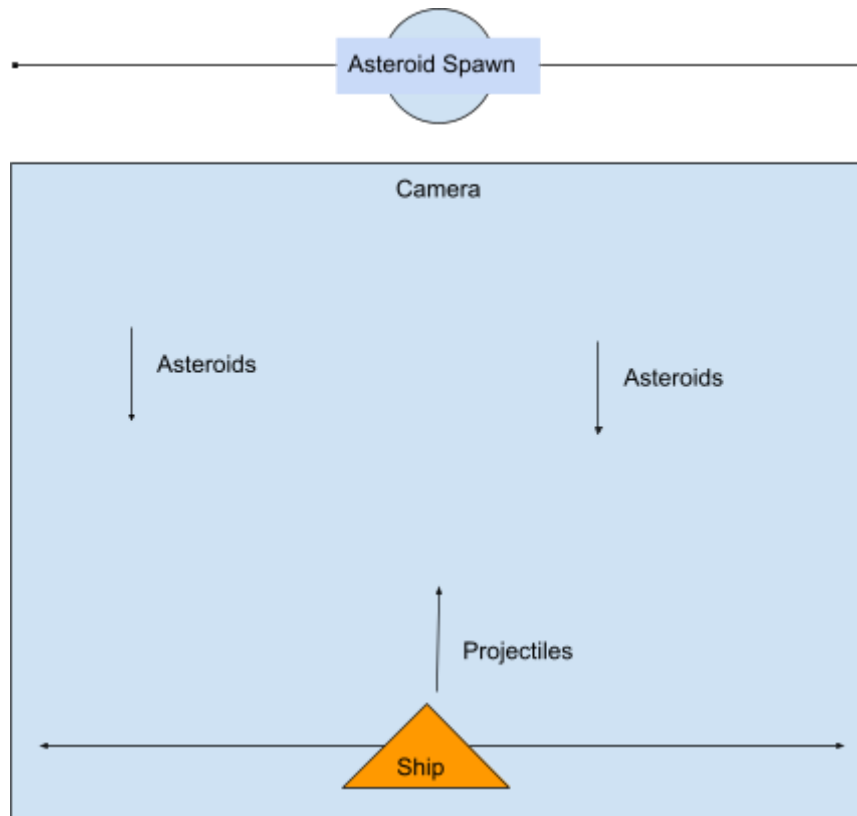


# Shoot 'em up

In this exercise you will assemble a very basic shoot 'em up game from the given assets. All scripts, sprites and sound files you need can be found in the project.





Game Layout, box resembles camera view.

The goal of the game is to shoot as many asteroids as possible. Asteroids give scores when shot and when a given amount of score is reached, the game is won. When an asteroid hits the player's ship the player loses.

*Find additional tips on GameObjects, Prefabs, Unity Physics etc. at the end of this document.*

## Part 1: Creating the project, the scene and importing assets

We will do this together, follow along in the class. For people at home:

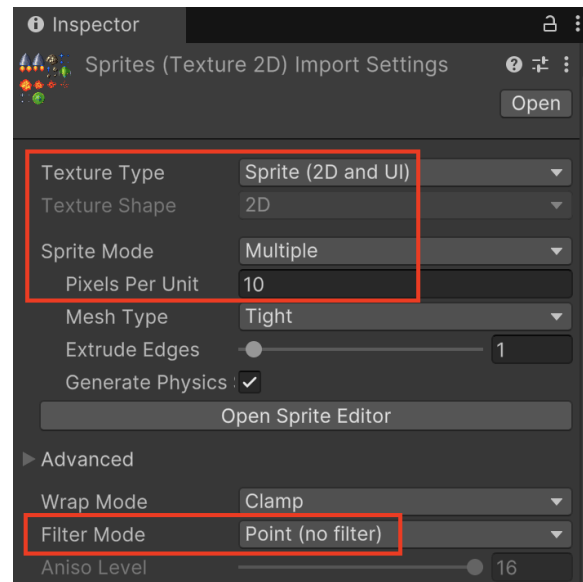
1. Create a new Unity 2D project with the name "Shootemup".
2. Import the assets found on canvas by extracting the zip "Shootemup\_Assets.zip" and dragging the folders into the Project window (or copying the folders into the "Assets" folder of your project in the explorer).
3. In the "Assets/Scenes" folder there is a scene file called "SampleScene" created by default by Unity. Rename that to "Game", open and save it.

## Part 2: Setting up the sprites for Pixel Art

We have all our sprites in 1 image (Sprites.png). We need to tell Unity to use the respective part of that image for each object. It also is a low-res 64x64 Pixel-Art image. That means we need to tell Unity to render these sprites as Pixel-Art (Without filtering).

### Importing pixel art images

1. Click on Sprites.png in the project window to open the import settings to the right (inspector)
2. Set up **Sprite Mode** to “Multiple” (many sprites in one image)
3. Adjust the **Pixels Per Unit**. How big are the images in the Unity scene? Here 10 image pixels will span one unit in a Unity scene.
4. Set **Filter Mode**. Point means no smoothing/filtering of the pixels. Perfect for Pixel Art.



Next hit: [Open Sprite Editor](#) to cut out our sprites.

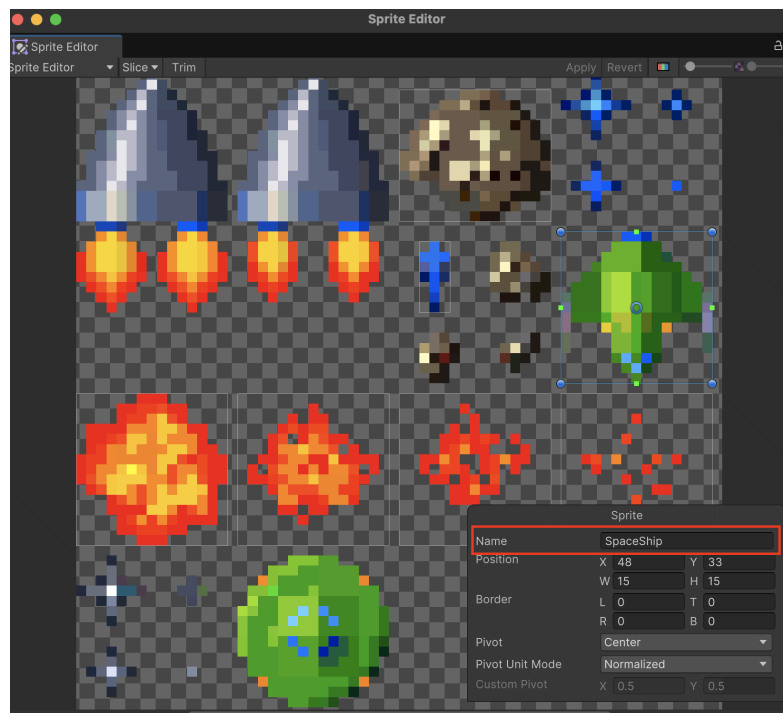
### Cutting out sprites in the Sprite Editor

Click and drag rectangles to create new sprites.


Give the sprite a name (see red box)

Cut out sprites for

1. Spaceship
2. Projectile
3. Asteroid
4. 4 Explosions.



After cutting the sprites, hit apply or close the editor.

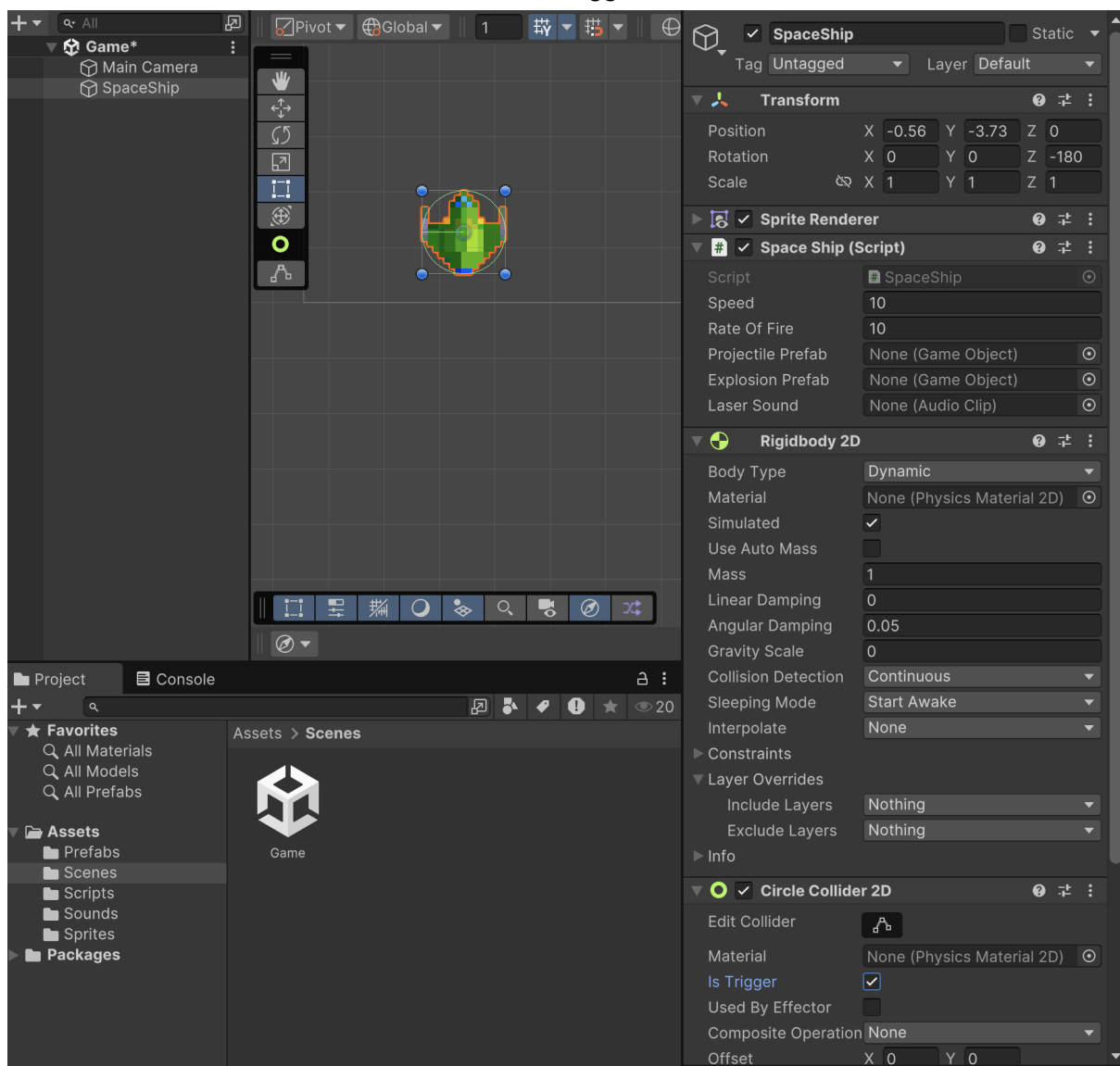
In your project window, your Sprites.png should now have the respective sprites (click )




## Part 3: The Spaceship

We will start by creating the spaceship gameobject in our Game scene.

1. Simply drag the Spaceship sprite into the scene window to create the GameObject.
2. Name the game object "Spaceship"
3. Attach the respective SpaceShip script as a component.
4. Also add a Rigidbody2D and a CircleCollider2D.
5. Make sure the CircleCollider2D has "IsTrigger" checked.

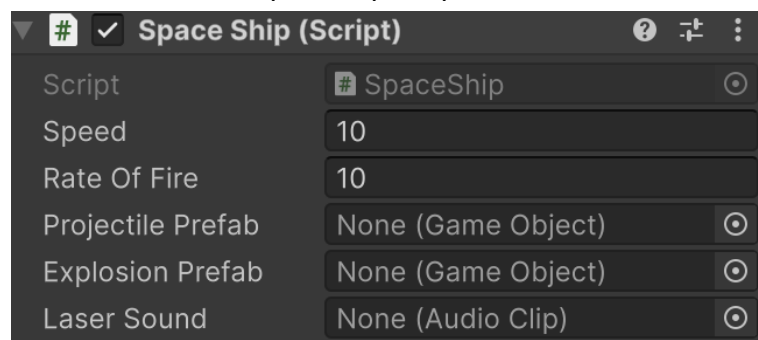


Hit the "Play" button:  You can now move the spaceship with the arrow keys!

## Part 4: Prefabs

Prefabs are blueprints for objects that are created on the fly during gameplay. In the Shoot Em Up we fire lasers. This means we need to clone or “instantiate” copies of a blueprint that tells Unity what a projectile looks like.

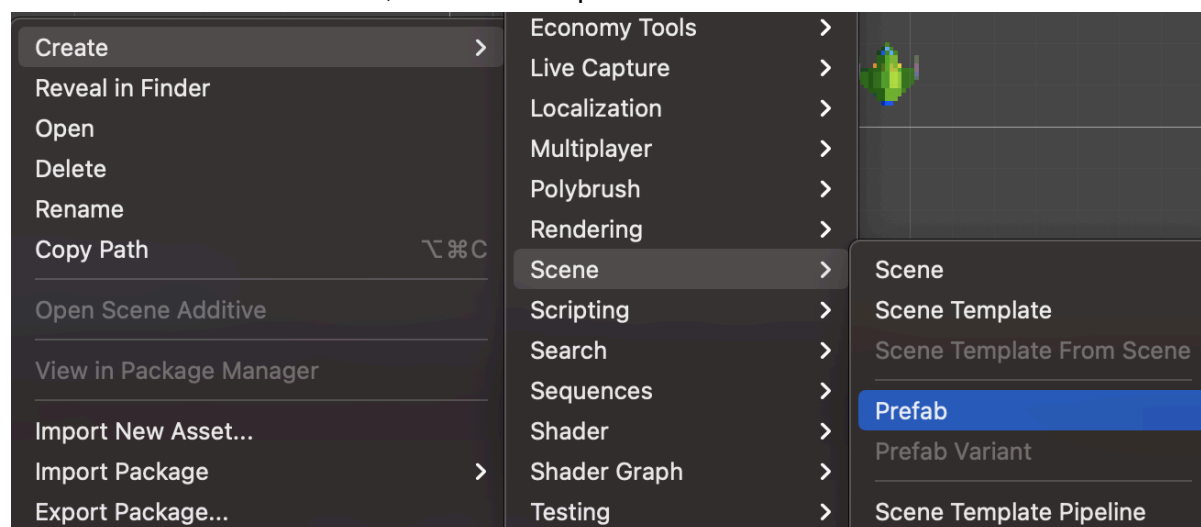
You can see in the spaceship script, that it wants to have a set of prefabs assigned:



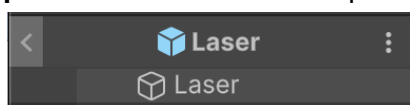
The spaceship needs to know about the Projectiles (to fire) and the Explosion (to explode), so it can create copies of those prefabs during gameplay. It also needs a sound to play when we fire the lasers.

### Let's create the projectiles first:

1. Create a folder “Prefabs” (if you haven’t already: Right-click->Create->Folder)
2. In the “Prefabs” folder, create a new prefab:



3. Name your new prefab: “Projectile” or “Laser”.
4. Double click to open the prefab editor (notice blue background, this means you are in **prefab edit mode**. To exit press this little arrow in the top right corner:



## Set up your projectile prefab

Quite a few Components this needs. You can always click “**Add Component**” and use the search bar to look for the component you need.

### Sprite Renderer

Drag in your Laser/Projectile sprite.

### Projectile Script

Scripts are like custom scriptable components giving objects their logic. Set up Speed and LifeTime

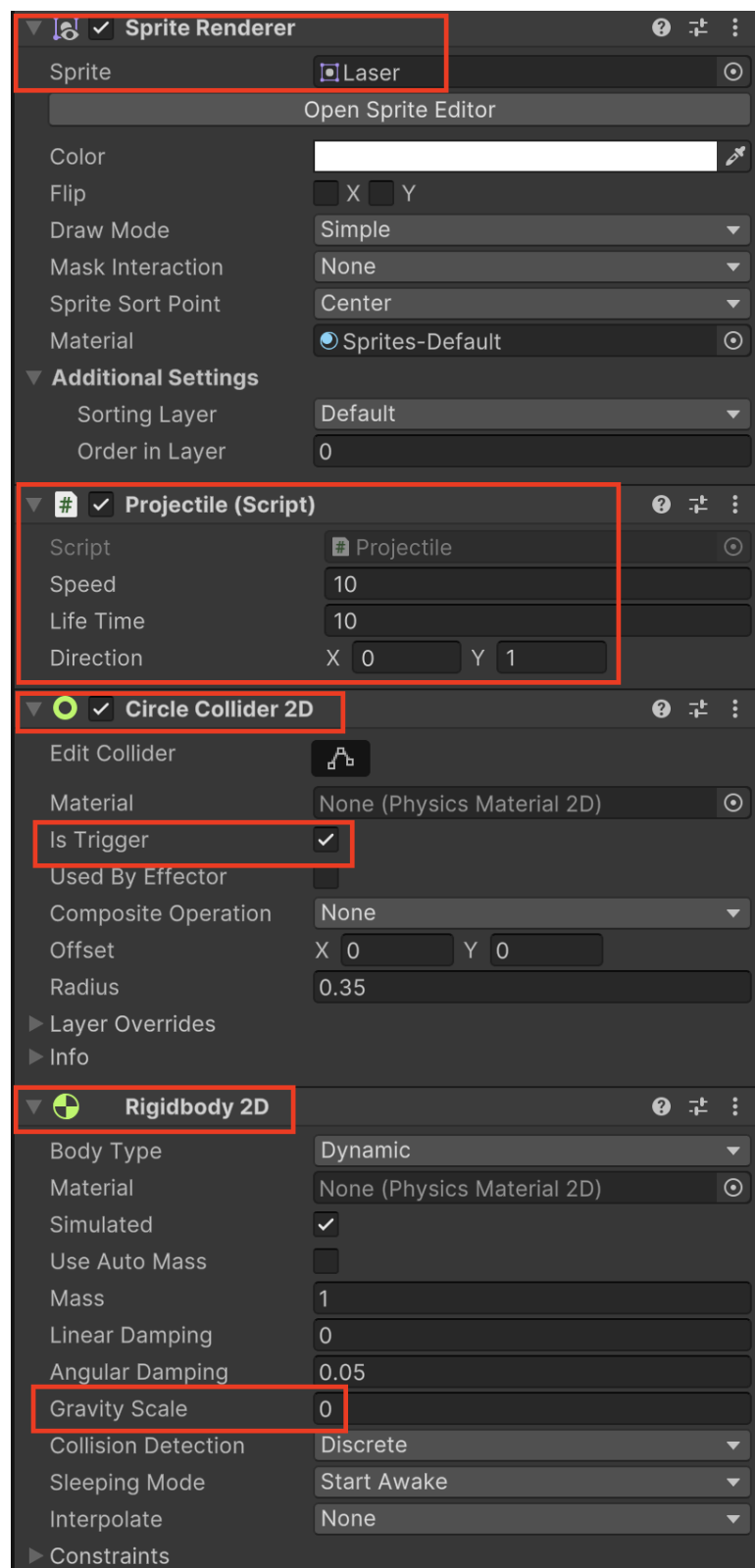
### Circle Collider 2D

We don’t need Newtonian physics and just check for overlaps in the script. So make sure Is Trigger is checked.

### Rigidbody 2D

To detect overlaps between two objects, at least one of them needs a Rigidbody2D. By rule of thumb:  
**ALL MOVING OBJECTS WITH COLLIDERS SHOULD ALWAYS HAVE A RIGIDBODY.**  
Sorry for the all caps, but this is important to remember.

Set Gravity Scale to 0, we have no 2D gravity in a top-down game.



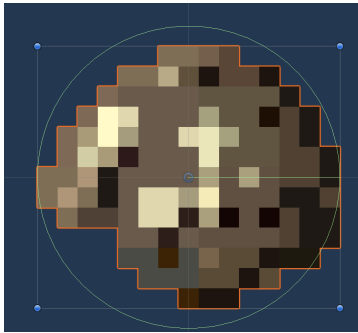
Now save and exit prefab mode (little arrow in the top left corner):



Add your new **projectile prefab AND the laser sound** (see sounds folder) to the SpaceShip script on your Spaceship GameObject. Press play and Fire Away (space bar)!

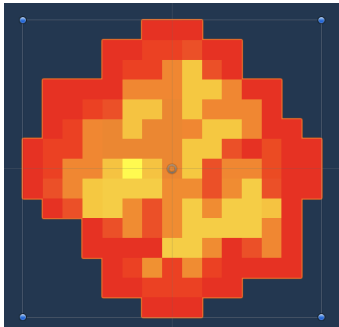
## Set up your asteroid and explosions prefab

We also need Prefabs for all other objects created during gameplay: Asteroid and explosion.



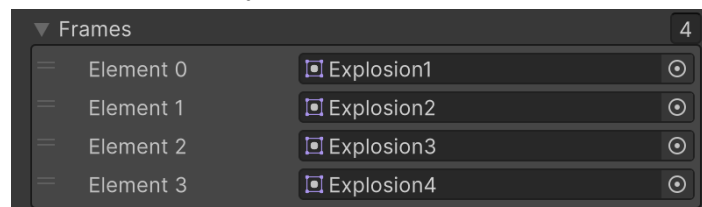
**Just like the projectiles set up a new Prefab for the Asteroid:**

1. Create a new prefab, call it "Asteroid"
2. Add SpriteRenderer with Asteroid sprite.
3. Add Asteroid script (set up values after you have explosions set up)
4. Add CircleCollider2D (check IsTrigger)
5. Add Rigidbody2D ( set Gravity scale to 0)



**Repeat one more time for explosions.** Explosions are a special effect GameObject, so they have purely visual purposes. **NO collisions and NO rigidbodies** here!

1. Create a new prefab, call it "Explosion"
2. Add SpriteRenderer with the first Explosion sprite.
3. Add Explosion script. The script will flip through the other explosion sprites (like a flipbook animation) and destroy the object afterwards. It needs to know about all explosion sprites
  - a. The "Frames" array in the Explosion script needs the sprite images:



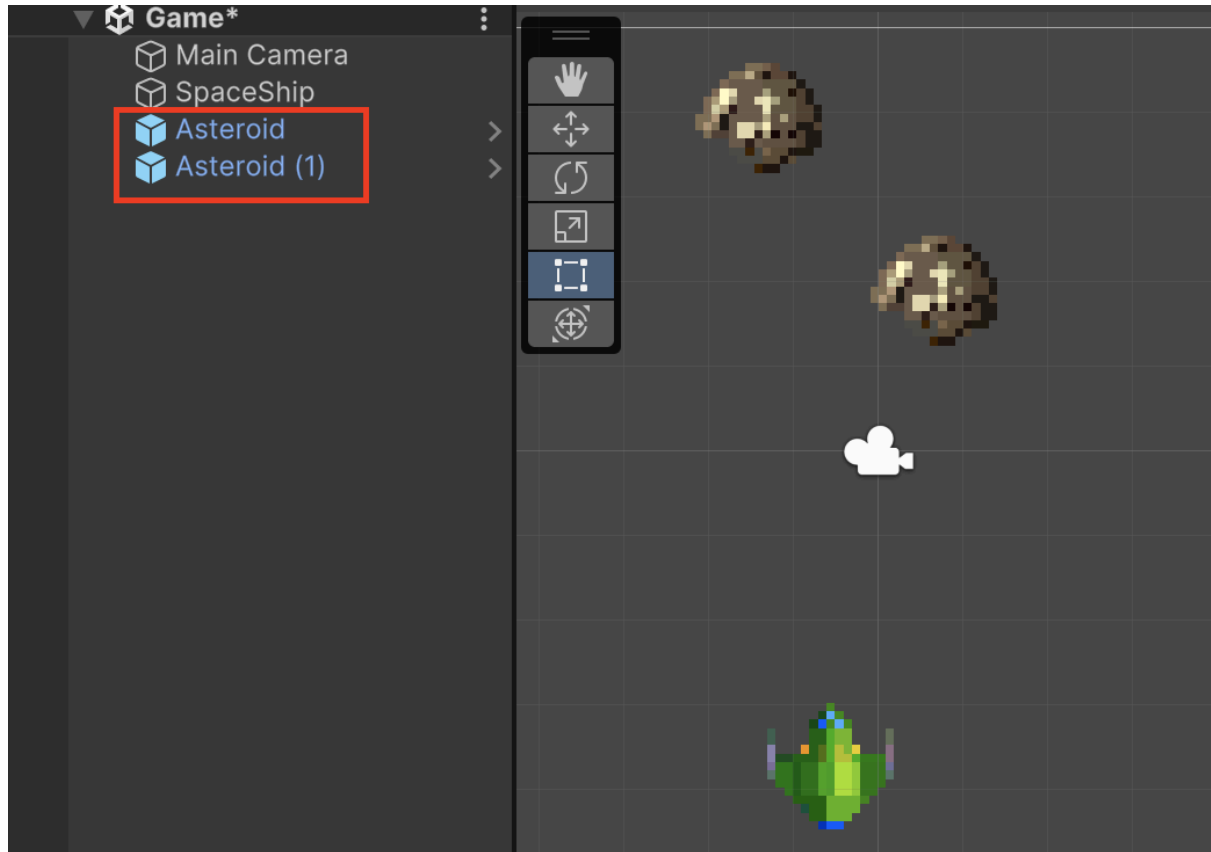
## Control your setup

Now check for all your scripts on Spaceship, and the Asteroid, Projectile and Explosion Prefabs that all the values have been assigned.

## Part 5: Test your game

Before we go to set up the Asteroid spawner, we want to test if things work:

Drag in two Asteroid Prefabs into your scene (just like with a sprite, but drag the prefab file!)



The blue box  means the Asteroids are copies of a prefab.

When pressing play, now all this should work:

1. Move the spaceship with the arrow keys.
2. Fire the laser with spacebar.
3. Lasers move upwards.
4. Asteroids move downwards and spin.
5. Laser overlaps with asteroids -> Explode!
6. Asteroid overlaps with spaceship -> Explode!

If anything does not work, check the console for errors:

 **UnassignedReferenceException: The variable laserSound of SpaceShip has not been assigned.**

Maybe you forgot to assign something to one of the scripts? The error message will tell you.

Or you could have forgotten to add Colliders/Rigidbody or forgot to check IsTrigger.

Or ask me for help:~)

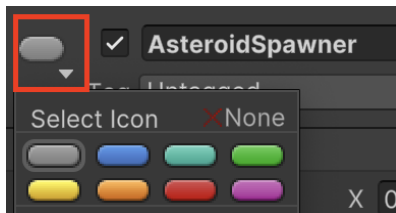


## Part 6: Asteroid Spawner

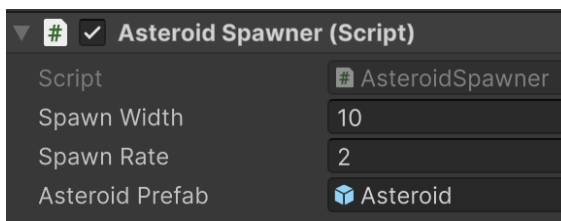
The spawner is our first object without a sprite. It is an “empty” with only a script attached that pumps our asteroids for us to shoot.

The spawner is not a prefab, since there is only one (like the spaceship).

1. Create a new “Empty” in the scene: right-click->create empty
2. Name it “AsteroidSpawner”
3. Since it has no graphics, the spawner is invisible in the scene. We can’t tell where the object is. But we can give it an icon to fix that (to the top right in the inspector):



4. Add the AsteroidSpawner script to it.



5. Move the spawner to the top of the scene, above what the camera can see:



**Great job you are done! Enjoy exploding those asteroids!**

# Bonus Goals (Optional)

## Part 4: Modifying the scripts.

1. Modify the asteroids to not all move at the same speed. Set the `speed` variable to be private and add two new public variables, `minSpeed` and `maxSpeed`. You can use `Random.Range()` in the `Awake()` or `Start()` function to assign a random value between `minSpeed` and `maxSpeed` to the asteroid when it is created.
2. Make the asteroids move in random directions by assigning random values to the x and y components of `direction` the same way. What range of directions makes sense?
3. `FireTheLasers()` on the `SpaceShip` script is used to spawn a new projectile when the fire input is given. Modify the function so it spawns multiple projectiles that travel in an arc outwards from the spaceship. You will need to modify the `direction` variable of the `Projectile` script. `FireTheLasers()` already grabs a reference to the script on the new object, but does not use it yet.
4. Create a `Bomb()` function on the `Spaceship` that kills all asteroids in the scene when the ctrl key is pressed. In the `Update()` function you can check if the ctrl key was pressed this frame using `Input.GetKeyDown(KeyCode.leftControl)`. In the `Bomb()` function use `GameObject.FindObjectsWithTag("Asteroid")` to find all objects with that tag. Use a `foreach` loop to loop over the resulting array and use `GetComponent<Asteroid>()` to get the `Asteroid` script on all found *GameObjects*. Then call `OnHit()` on them to destroy them.

# Appendix, additional information on Unity, Physics and Collision Detection

## 2D Game Physics in Unity

It's easy to use the physics system in Unity in order to determine collisions and create movement. Be careful to only use the 2D components like *Rigidbody2D* and *BoxCollider2D*, when you are creating a 2D game, since they are not compatible with the 3D equivalents *Rigidbody* and *BoxCollider*.

## Colliders and Rigidbodies

To handle collision and trigger notifications in scripts (execute the, `OnCollisionEnter2D(Collision2D col)` or `OnTriggerEnter2D(Collider2D other)` or similar functions) Unity needs *GameObjects* set up in a very specific way.

The *GameObjects* participating in the collision both need to have any type of *Collider2D* (circle, box or polygon) attached. Furthermore, one or both of them need a *Rigidbody2D* component. A *Rigidbody* is a basic physically simulated body, meaning it receives forces, gravity and can collide with other objects.

This *Rigidbody2D* needs the “Simulated” box checked to, well, be simulated, aka move.

When objects collide (overlap), if one or both of the two *Colliders* has the “IsTrigger” box checked, `OnTrigger...` functions will be called. If none of the *Colliders* are Triggers, the `OnCollision...` functions will be called.

## Tags

`OnTriggerEnter2D()` on the *Projectile* checks for the other *GameObjects Tag*, to see if it collided with the correct object. This required the asteroid object to have the “Asteroid” tag. When setting up the asteroid, add it to the already existing tags in the inspector by clicking “Add Tag” under “Tag”.